C Notebook

Hints and Queries

Compiled and Linked by Lee Calcraft

MISSING ERROR FLAG

by Dave Appleby

In both ANSI C 3 and 4, the no error flag used for error windows is missing from the wimp.h header. It can be added to the wimp_errflags enum in wimp.h as:

```
wimp_ENOERROR=16
```

or #defined as required, enabling wimp_reporterror() to be used as an information window as well as for error reporting.

MYSTERIOUS DISAPPEARANCES

by David Pilling

Sometimes application programs will vanish, with no stack backtrace or error box. One cause of this is the C file library.

If your application's WimpSlot is only just big enough, then it seems that opening files causes the C library to run out of space. The problems only show up when files are closed. At this point the C library prints (in text mode) that it has run out of memory, and kills the task stone dead. So the tip is, if you suffer from disappearing programs, press F12, and open a spool file (*spool trace), then go through the steps that cause the trouble. Close the spool file (with *spool), and load it into Edit and look for any hidden messages from the C library (usually bracketed with a couple of **s). The spool file will trap all VDU output, and thus the helpful messages which were printed somewhere off screen.

POINTER TROUBLE

by Lee Calcraft

The information given about RISC_OSLib functions in the ANSI C manuals is excessively concise, and can lead to slip-ups on the user's part. For example, you need to take care when a function requires a pointer to an empty structure, into which it will put a result. Thus with:

```
wimp_get_caret_pos(wimp_caretstr*)
```

it is no good simply defining a suitable (wimp_caretstr *) pointer, and passing this - your program would soon crash. You must define a wimp_caretstr structure, and then pass the pointer to this to the function.

As a guide, if RISC_OSLib is going to provide memory for a particular structure, the function concerned usually returns the structure, as with dbox_new() for example, which returns a dbox.

IMPROVED TRACE MACROS

by Chris Wilmott

The following macros provide a better trace facility than that offered in ANSI C. They display formatted debugging output at a particular screen line on the Desktop.

To use the feature, include the macros at the start of a source, or in a header, then use #define TRACE 1. Otherwise #define TRACE 0, in which case no trace code is compiled.

As an example, the following:

```
show3(8,"Mouse x=%d y=%d    ",\
            d->m.x,d->m.y);
```

might be used to display the mouse co-ordinates on line 8 of the screen. Use show1() if you have one parameter to pass to printf(), show2() if you have two or show3() if you have 3.

```
#include "bbc.h"
#include <stdio.h>

#if TRACE
 #define tron(a) bbc_vdu(4); bbc_vdu(26);\
                 bbc_vduq(31,0,a)
 #define troff(void) bbc_vdu(5)
 #define show1(a,b) {tron(a); printf(b);\
                 troff();}
 #define show2(a,b,c) {tron(a);\
```

```
                printf(b,c); troff();}
#define show3(a,b,c,d) {tron(a);\
                printf(b,c,d); troff();}
#else
#define show1(a,b)
#define show2(a,b,c)
#define show3(a,b,c,d)
#endif
```

Please send us your C hints - all published hints will be paid for.